

# DNA self-assembled parallel computer architectures

Chris Dwyer<sup>1</sup>, John Poulton<sup>2,3</sup>, Russell Taylor<sup>4</sup> and Leandra Vicci<sup>4</sup>

<sup>1</sup> Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA

<sup>2</sup> Rambus Incorporated, Los Altos, CA, USA

<sup>3</sup> Rambus Incorporated, Chapel Hill, NC, USA

<sup>4</sup> Department of Computer Science, University of North Carolina, Chapel Hill, NC 27599, USA

E-mail: [dwyer@ece.duke.edu](mailto:dwyer@ece.duke.edu)

Received 4 August 2004, in final form 14 September 2004

Published 28 October 2004

Online at [stacks.iop.org/Nano/15/1688](http://stacks.iop.org/Nano/15/1688)

doi:10.1088/0957-4484/15/11/055

## Abstract

New varieties of computer architectures, capable of solving highly demanding computational problems, are enabled by the large manufacturing scale expected from self-assembling circuit fabrication ( $10^{12}$ – $10^{19}$  devices). However, these fabrication processes are in their infancy and even at maturity are expected to incur heavy yield penalties compared to conventional silicon technologies. To retain the advantages of this manufacturing scale, new architectures must efficiently use large collections of very simple circuits. This paper describes two such architectures that are enabled by self-assembly and examines their performance.

(Some figures in this article are in colour only in the electronic version)

## 1. Introduction

The emergence of new device technologies is widely speculated to supplant the diminishing returns from the continued scaling of conventional silicon technology as we approach the ‘red brick’ wall identified by the International Technology Roadmap for Semiconductors [1, 2]. There is a wide array of new devices and architectures that explore the application of nanoscale research to computer design [3]. Some of these emerging computer designs represent fundamental shifts in the methods used to compute, and some are clever applications of conventional logic design to novel nanoscale device technologies.

The self-assembling computer architectures described in this paper represent two ends of a computational spectrum. One end of this spectrum includes the design space used by conventional computer designs to execute stored programs at ‘run-time’ using intricate networks of electrical circuitry. The other end of this spectrum includes a design space used by DNA computing schemes to perform a computation at ‘assembly-time’ using the orchestration of DNA molecules and an intricate series of biochemical processing steps. The continuity of this spectrum embodies the trade-offs between run-time performance and assembly-time complexity.

The first architecture, the decoupled array multi-processor (DAMP), is similar to a single-instruction, multiple-data (SIMD) design and relies heavily on run-time computation and is therefore on the run-time side of the computational spectrum. The second architecture, an oracle, is similar to a content-addressable memory with the important distinction that this memory *self-assembles* itself and its contents during its fabrication and is on the assembly-time side of the computational spectrum.

Both architectures are enabled by a guided self-assembly process that uses DNA hybridization to precisely structure nanoscale circuitry. Like many other emerging technologies, this process is being developed as a replacement or supplement to the photolithographic patterning processes used with conventional silicon technologies. The designs presented here are simple enough to be realized by self-assembly yet powerful enough to exploit the vast manufacturing scale promised by this new class of fabrication.

### *Self-assembly*

Pioneering work on the use of DNA molecules as structural components, first introduced by Seeman [4, 5], has paved the

way for methods that use DNA as a scaffolding or assembly agent [6–10]. This inherently bottom-up approach is supported by work developing compatible nanoelectronic components that are amenable to DNA-guided self-assembly [11, 12].

On-going research into DNA-guided self-assembly will clarify the details of how to efficiently assemble nanoelectronic components. In the meantime, we base our discussion of computer architecture on recent work in self-assembly that has begun to discern important challenges. The basic premise is that hybridization between complementary DNA strands can form complex networks of electronic components in vast quantities. These networks can be designed to function as logic blocks and have been simulated to estimate their electrical performance [13, 14].

However, even with a per assembly-event yield of 98%, large structures are likely to form only a fraction of the time. Therefore, it is important to rethink the way large systems are designed (and emphasize smaller, simpler components) if the great potential of self-assembly is to be realized.

### Nanoscale computer design

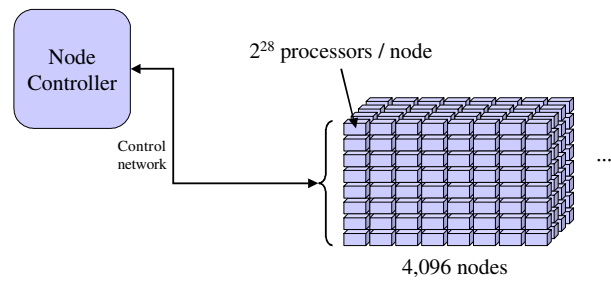
A great deal has been studied in terms of how computer systems might migrate from traditional silicon technology to nanoscale and self-assembling technologies. Defect tolerance at the nanoscale is an architectural issue that deals with the change in fabrication properties and computing systems from conventional silicon technology. The Teramac computer was an early adopter of reconfigurable defect tolerance [15] and other work has demonstrated the advantages of reconfiguration over traditional redundancy [16, 17]. The application of nanoelectronics to parallel computing has emerged as a new area of focus between materials science, chemistry, physics, and computer design [18, 13, 19–22]. Alternative approaches to von Neuman computing have re-emerged as certain aspects of nanoelectronics mature. Cellular architectures employ local interactions between elements to compute [22–25]. This form of communication is expected to dominate at the nanoscale [26]. Reconfigurable array architectures are also being investigated to cope with the high defect rates expected of nanoelectronics [21, 27, 28]. The great promise of quantum computing has driven investigations into practical issues including architectures [29, 30] and communications [31].

Sections 2 and 3 describe two architectures that are enabled by developments in self-assembled nanoelectronics.

## 2. The decoupled array multi-processor

The early limitations of self-assembling technologies require designs to use small circuitry and little interaction or communication with the external (i.e., macroscopic) world. Single-bit, serial processing elements are well suited to such limitations. They require less circuitry than parallel multi-bit implementations and have simple interfacing requirements.

The particular active component and assembly process we consider here is theoretical and based on prior work in emerging device modelling [14, 32] and fabrication [33] of silicon rod surrounding gate (or ring-gated) field-effect transistors. The process, described in [34], uses DNA functionalized silicon rod FETs and precise control over



**Figure 1.** The node controller and processor node arrangement.

the type of rod (e.g., heavily doped for wires, N+, P+, or insulating) and the sequence of strands of DNA covalently bonded to each end. The DNA strands direct the ends of the rods to assemble based on their sequence as in mesoscale DNA self-assembly [35–37]. Post-process metallization after the DNA has assembled the silicon rods renders the strands conductive [38].

### 2.1. System overview

The decoupled array multi-processor (DAMP) is similar to a single-instruction multiple-data (SIMD) machine (e.g., the CM-2) with two important differences: the DAMP has no inter-processor communication, and many more processors ( $\sim 10^{12}$ ). The most significant of these differences is the lack of any communication hardware between processors because most efficient solutions to common parallel algorithms require inter-processor communication of some sort.

The processors in the DAMP have no way to communicate with each other (i.e., they are decoupled from each other) except through a shared control unit. This limited form of inter-processor communication limits the DAMP to embarrassingly parallel problems that involve little to no data sharing.

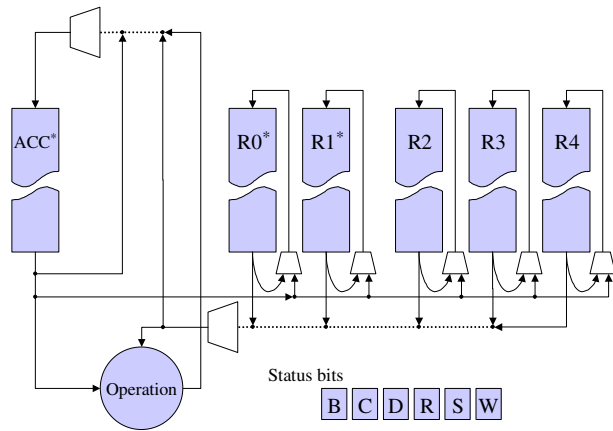
The magnitude of the number of processors in each machine type is also dramatically different from typical SIMD machines. With on the order of  $10^{12}$  processing elements the DAMP far exceeds typical processor counts. However, the complexity of any individual processor is greatly diminished with respect to the processors used in SIMD machines. The basic structure of the DAMP is illustrated in figure 1. The node controller sends control signals to each processor node in parallel. Each processor node can detect a processor-generated signal from an individual ‘ringer’ circuit embedded in each processor.

This reduced output capacity (from the processor’s perspective) is a conservative worst-case scenario to reduce the fabrication complexity of each processor.

### 2.2. Execution model

Figure 2 illustrates the basic execution model for a DAMP processor. Each processor has five 16-bit registers and conditionally executes the instruction stream depending on the value of its wait-status bit.

In this bit-serial design, the least significant bit (LSB) is the first bit to participate in each operation: the bit at the bottom of a register in figure 2. Through separate shift controls, the accumulator can shift independently from the R0–R4 registers



**Figure 2.** Processor diagram. ACC, R0, and R1 can be loaded with random bits.

enabling relative data shifts. The operational unit is a full-adder that can provide either the carry out or sum signal to the accumulator input. Each register R0–R4 can receive either its own LSB or the accumulator output as input during a shift. The six status bits can be used to implement a wide range of conditional operations [13].

The 16-bit accumulator, R0, and R1 have the ability to load a random constant that is probabilistically unique to each processor through the use of a random assembly event that selects either a one or zero for each bit in the register (e.g., by a conductor or insulator attaching randomly to a latch’s load line). The random constant is used as an index or a seed for selecting a portion of a problem space. At run-time each processor can supplement these assembly-time input bits with a counter or value from the node controller in the LSB positions.

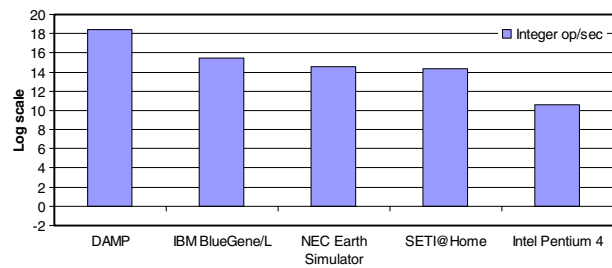
The DAMP processors ( $\sim 10^{12}$ ) can probabilistically evaluate any 40-bit input space with only one run of a program. The program instructs each processor to manipulate their random constants to produce an answer to a problem. If the answer is satisfactory (e.g., below some threshold) the processor with the answer can alert the node controller (with its ringer circuit) and a binary search over that node’s input space can begin. The search is complete when all bits of the random constant used by the target processor are determined.

The self-assembly fabrication technology used to build the DAMP demands that each processor be simple. This drives the use of bit-serial processors and a simple controller. As a consequence all instructions are software encoded without the need for microcode on each processor (e.g., like VLIW). A brief list of instructions that can be efficiently implemented by the DAMP, including the gate level implementation details, transistor level nanorod layout for the processing elements, and the behavioural simulation details, can be found in [13].

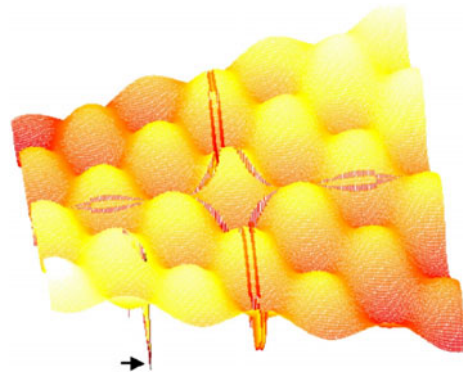
### 2.3. System operation

The large number of processors in the DAMP ( $\sim 10^{12}$ ) leads to a very high peak performance on 16-bit operands (e.g., local 16-bit adds) as illustrated in figure 3. However, peak performance is only a simple measure of usefulness. The real power of a computing machine comes from the problems it can solve.

The DAMP can be used to solve vast global optimization problems. Many science and engineering problems can be



**Figure 3.** Peak performance of several machines on 16-bit operands.



**Figure 4.** A constrained objective function with many local minima. The black arrow indicates the minimum for this region.

posed as global optimization problems that seek to find the largest or smallest value for an objective function over a domain. The challenge in solving these problems comes from the large number of variables and multiple local minima that deceive search algorithms. Consider the hypothetical objective function shown in figure 4. This function has many local minima and the global minimum, indicated by the black arrow, has a very narrow aperture for the search algorithm to find.

Stochastic global optimization is a method of sampling an objective function at random points in the problem space and comparing the results at each point. Since the time required to exhaustively search the problem space at a resolution sufficient to be useful is far too large, the best local minimum is chosen from local searches starting at a random set of starting locations. A new set of random points is selected that concentrates the search around the best-found solution. That is, the search continues but focuses on a few of the last-best answers. Typical calculations for each sample include the objective function and numerical derivatives (gradients) at the point. If the objective function has a well-behaved and computable gradient, this can be used as a local indicator of how to choose the next best solution since the objective changes along the gradient. This *gradient descent* approach is very sensitive to numerical instability because of how derivatives amplify high frequency changes in the objective function. The gradient descent approach is also very susceptible to entrapment by local minima.

Parallel pattern search (PPS) has emerged as a popular technique used to optimize difficult objective functions [39–42]. This method uses a search along each dimension of the problem space to find the global minimum. Each iteration of the search begins from the optimal point found in the last round

of evaluations. This technique has provable convergence to the minimum as long as certain rules are followed for adjusting the step size along each dimension and for how objective values are compared.

The DAMP can be used to solve continuous variable minimization problems that are much larger in dimensionality than those solvable today. The pseudo-code below demonstrates how the DAMP can be used with 32-bit fixed-point variable optimization problems.

```

For each processor node  $j$  ( $PN_j$ ),
  For each processor at node  $PN_j$ ,
     $k$ , ( $k$  is a random 28-bit
      integer)
    1:  $\Delta k = k \ll 4$ ;
    For 16 iterations,
      2: evaluate and verify  $C_x(x_k + d_j \cdot \Delta k)$ 
      3: evaluate  $y = F(x_k + d_j \cdot \Delta k)$ ;
      4: evaluate and verify  $C_y(y)$ 
      5: participate in MIN-
        QUERY( $y$ ,  $x_k + d_j \cdot \Delta k$ )
      6:  $\Delta k = \Delta k + 1$ ;

```

The vector  $x_k$  is the best-known solution after each step. The problem space is spanned by a positive spanning set<sup>5</sup>  $\mathbf{D}$ , where  $d_i$  is a unit vector from  $\mathbf{D}$  along the  $i$ th dimension of the problem space. The functions  $C_x(x_i)$  and  $C_y(y)$  are used to verify that the input and output vectors, respectively, satisfy the problem constraints.

The program is run at each processor node. Since there are  $2^{28}$  processors per node, the random number generated at each processor has only 28 bits of significance. This means that to cover a 32-bit random number space each processor must run the program 16 times with a new 4-bit low order value each time. The value,  $\Delta k$ , is simply incremented between loops. Each processor takes its  $\Delta k$  and uses it to compute a new input vector. The particular dimension that the processor searches ( $d_i$ ) is specific to the processor node. The new input vector is checked against the input constraints and if they are satisfied the function ( $F$ ) is evaluated. The output from the objective function is checked against the output constraints and if they are satisfied the processor participates in a minimization query, or MIN-QUERY. This query is conducted by each processor node and searches, bit by bit, for the smallest objective function value found by any of its processors.

In addition to the details of the MIN-QUERY algorithm and implementation details of each instruction, a practical application of this method to the thermal intercept problem and performance results can be found in [13].

### 3. Oracles

Similar to the DAMP, oracles are assembled using electrically active components (silicon rods) and DNA (metallized during post-processing) to form large arrays of simple circuitry.

<sup>5</sup> A positive spanning set is a set of vectors that can be combined using non-negative scalars to form all possible vectors in a constrained space.

**Table 1.** Full-adder truth table.

$A$	$B$	$C_i$	$S$	$C_o$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

However, unlike the DAMP, oracles are designed to solve large portions of a particular problem during their fabrication rather than relying on purely run-time computation. That is, the oracles are a hybrid between the general-purpose electrical designs described in section 2 and electrical circuitry inspired by DNA computing. This section describes the architecture of an oracle and two simple designs.

#### 3.1. System overview

An oracle is designed to solve a problem space during its assembly. This architecture is enabled by DNA-guided self-assembly because DNA hybridization can enforce well-defined sets of pairing rules. So far, these rules have been used here to define the geometric structure of circuitry (e.g., in the DAMP) but as in DNA computing, the rules can also be used to compute a result.

Abstractly, an oracle contains a large number of question and answer pairs. Questions are posed to the machine and it generates a response if the question is contained in any of the oracle's question/answer pairs. In this fashion, the oracle is like a large content-addressable memory (CAM) that has been preloaded with the answers to a certain problem. An oracle differs from a CAM by the method the question and answer pairs are entered into the machine. To fully cover an input space of  $k$ -bits, the CAM requires  $O(2^k)$  steps to load the answers (each of which must be computed). Each address is a question represented by up to  $k$ -bits with its associated answer—just like a lookup table. In comparison, the oracle requires  $O(k)$  steps to assemble and no run-time loading steps. The answers are determined by the manner in which the oracle is assembled. The self-assembly of each question and answer pair provides the oracle with the answers (with a high probability but not a certainty that a given question and answer pair will exist within the oracle). If a particular question and answer pair did not form during the oracle's assembly then the oracle cannot solve that instance of the problem.

#### 3.2. A simple oracle

A simple example of an oracle is the addition oracle (not useful in itself, but illustrative.) The addition oracle has a simply defined problem and a brief functional description, and performs all calculations at assembly time.

Table 1 lists the entries in the truth table for a full-adder. The addition oracle will be assembled so that it incorporates many of the possible combinations of the truth table entries. Each line from the truth table represents an instance of an addition problem. A ripple-carry adder solves

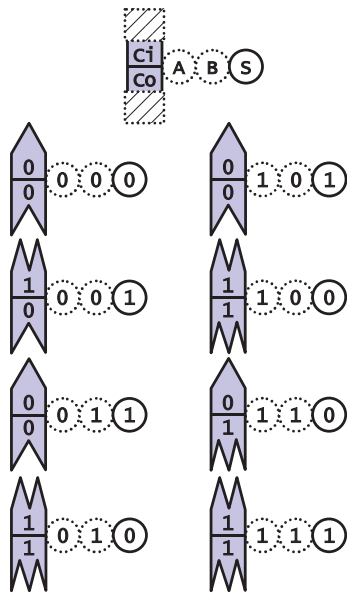


Figure 5. Assembly tiles for the addition oracle.

multi-bit addition problems by chaining the carry-out from one full-adder to the carry-in of the next full-adder. In a similar way the addition oracle will chain carry-outs to carry-ins, but at assembly-time rather than at run-time.

### 3.3. Implementation

Each line in the truth table is converted to a ‘tile’ that represents a particular input and output combination as in [43]. The difference in this work is that each ‘tile’ is implemented by the self-assembly of nanoelectronic components and is electrically active. Many of the challenges in fabricating large networks from DNA will remain in fabricating the DNA scaffolded nanoelectronic circuitry considered here. However, progress is being made in this area [44] and discoveries in the fundamental properties of DNA self-assembly will apply to this theory directly.

The tile conversion method is similar to the way a carry-select adder speculatively pre-computes a carry pattern and at run-time selects the proper carry path, with the exception that the oracle pre-computes the entire carry path. The tiles that correspond to table 1 are shown in figure 5. Each tile has on its left side a carry input and an output. In this case, the top portion of the tile is the carry-in and the bottom portion the carry-out. The carries are depicted in such a way that they fit together like the pieces of a jigsaw puzzle.

The iterative nature of the function (e.g., output from step  $i$  produces input for step  $i + 1$ ) allows strings of these tiles to implement an instance of the function’s evaluation. For example, figure 6 illustrates a simple 4-bit string made from the tiles in figure 5. This particular example is an instance of the addition function for ‘ $3 + 5 = 8$ ’. The shape of the carries on each tile dictates how the string is formed. Valid strings must match each carry-out with the corresponding carry-in. In this fashion, the tiles perform an assembly-time computation as they form valid strings. They assemble only into valid solutions for addition because the carries must match at each stage.

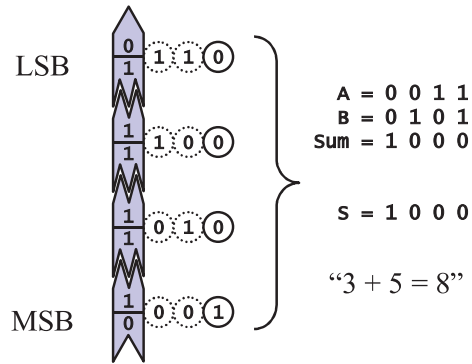


Figure 6. A 4-bit instance of the addition function. The carry-in and carry-out rules determine valid strings.

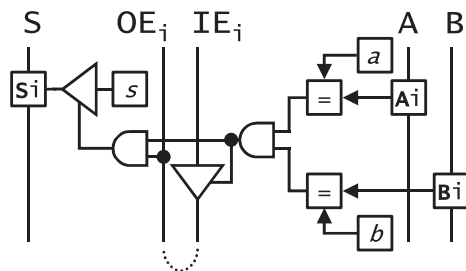
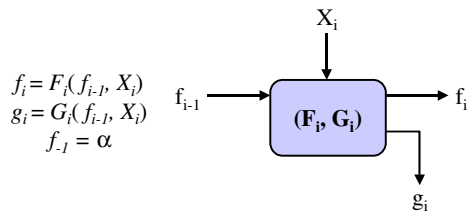


Figure 7. Circuit for an addition tile. The  $a$ ,  $b$ , and  $s$  bits are constants assembled into a particular tile.

A complete addition oracle is the collection of all possible  $N$ -bit strings of tiles. Each string represents one particular input and output combination, for example the ‘ $3 + 5 = 8$ ’ string shown in figure 6. In this case, the string will respond with ‘8’ to the question ‘What is  $3+5$ ?’. For all other questions the string will be silent.

Again, the addition oracle is simply an illustration of an oracle rather than an exemplar of its usefulness. The circuit complexity of each string is determined by the circuitry needed to read the string and respond to queries. A possible circuit for the addition oracle is shown in figure 7.

The  $A$ ,  $B$ , and  $S$  signals illustrated in figure 7 carry the input query and response bits, respectively, for each tile. The  $OE_i$  and  $IE_i$  signals are the output and input enable signals, respectively, that coordinate the individual tiles in a string so that the string responds to a query if and only if *all* tiles in the string match the input query. The input enable signal is passed downward along the string and at the very last tile is reflected upward as the output enable signal. Each tile can interrupt the input enable signal depending on the value of the current query, or the latched  $A_i$  and  $B_i$  input signals. Input queries are serially shifted into all strings (i.e., the circuits that implement each string) simultaneously. When the  $A$  and  $B$  values match the particular inputs of a string, all the tiles latch their sum values into the  $S_i$  latches. The only strings that respond to the query are those that have successfully reflected the input enable signal to their output enable line. The output enable signal can be used to trigger a ringer circuit that creates an oscillating signal that can be detected by an external receiver. This method is useful for problems that require only a single bit of output (e.g., NP-complete problems). Alternatively, the output enable signal can be used, as shown in figure 7, to load



**Figure 8.** Problem expression solvable by an oracle.

the sum bit into a D-latch that can be shifted downward along the string to a ringer at the bottom that responds to the shift-out from the string.

### 3.4. Generalization of the oracle

The addition oracle serves as a simple model for other oracle designs. Carry chains are simple input and output constraints that can be generalized to include more complex relationships.

An oracle can solve a problem that can be expressed using the form illustrated in figure 8. The functions  $F_i$  and  $G_i$  take the  $f_{i-1}$  and  $X_i$  inputs and generate the  $f_i$  and  $g_i$  outputs, respectively.

Each input ( $f_{i-1}$  and  $X_i$ ) and output ( $f_i$  and  $g_i$ ) are bit vectors. To aid in initializing the system,  $f_{-1}$  is assumed to be  $\alpha$ , which is a constant defined at assembly-time.

Equations (1)–(3) describe the addition oracle using the form illustrated in figure 8. These equations are derived from the truth table for addition, shown in table 1. The input vector  $\mathbf{X}$  has two elements,  $\mathbf{A}$  and  $\mathbf{B}$ , that represent the input operands. Equation (1) is the carry-out bit, and equation (2) is the sum bit.

$$F_i(f_{i-1}, X_i) = f_{i-1} \cdot (X_i[A] + X_i[B]) + (X_i[A] \cdot X_i[B]) \quad (1)$$

$$G_i(f_{i-1}, X_i) = \overline{f_{i-1}} \cdot (\overline{X_i[A]} \cdot X_i[B] + X_i[A] \cdot \overline{X_i[B]}) + f_{i-1} \cdot (\overline{X_i[A]} \cdot \overline{X_i[B]} + X_i[A] \cdot X_i[B]) \quad (2)$$

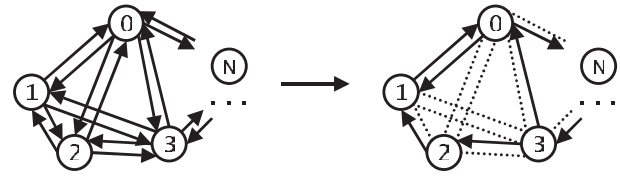
$$\alpha = 0. \quad (3)$$

### 3.5. Hamiltonian path oracle

The Hamiltonian path (HAM-PATH) problem is NP-complete and represents what is considered to be an intractable problem. The problem consists of finding a path through a connected graph of nodes that visits each node exactly once.

The HAM-PATH oracle computes all paths through a fully connected graph at assembly-time in a manner very similar to the way Adleman solved the HAM-PATH problem using DNA [45]. The difference between the oracle and Adleman's approach is that the oracle solves the problem for any instance of the problem (with fewer than a fixed number of nodes).

Adleman's solution encodes each edge in a graph as a DNA fragment that has two 'sticky' ends representing the starting and ending nodes of the edge. Each node in the graph is allocated a sequence of DNA, and any edge that starts at that node will use this sequence on one end. The other sticky end of the DNA fragment uses the complement of the DNA sequence assigned to the ending node. All of the fragments are mixed together and form strings of edges (in the form of DNA fragments) that represent feasible paths through the graph. Since Hamiltonian paths visit each node once, only



**Figure 9.** The fully connected graph on the left is collapsed to a particular graph on the right. The dashed lines represent deleted edges.

strings with as many edges as there are nodes in the graph are feasible Hamiltonian paths. All other strings are discarded (using biochemical techniques). Cycles in the graph need special treatment [1]. The entire process takes on the order of weeks from start to finish.

The way the HAM-PATH oracle solves all instances of the problem is by solving the problem for a fully connected graph and then discarding solutions at run-time based upon a particular input graph. Paths from the fully connected graph that do not appear in the problem instance are deleted at run-time. This idea is illustrated in figure 9.

Like the addition oracle, the HAM-PATH oracle uses random strings of tiles to perform an assembly-time computation. The addition oracle formed all  $N$ -bit sums at assembly time. Likewise, the HAM-PATH oracle forms all paths through the fully connected graph. At run-time the HAM-PATH oracle selects the edges that exist in the current problem instance one or more computing elements within the HAM-PATH oracle responds (electrically) to indicate that a Hamiltonian path exists through the graph if and only if it has a solution.

The design of the circuitry for each HAM-PATH tile is more complicated than the addition oracle tiles because they need to remove nodes from a set and then respond to selected graph edges. However, the generalization in section 3.4 still holds and a more detailed account of this design and its circuitry can be found in [13]. The circuit level simulation results of this architecture estimate a run-time cycle period of  $\sim 10 \mu\text{s}$  per graph instance to compute the optimal path. This is approximately 40 000 times faster than the estimated earth simulator performance (200 ms).

## 4. Conclusion

The challenges that self-assembly present to system design demand alternative approaches to large-scale computing. The dramatic change in the fabrication scale ( $10^{12}$  devices) and severe constraints on circuit size must be balanced to exploit the advantages of DNA self-assembly. The two architectures we have proposed are enabled by self-assembly because they rely on its unique properties (i.e., scale and rule-based assembly) and can solve more complex problems than possible today (e.g., the 15-node HAM-PATH) with promising direction toward more practical applications (e.g., large global optimization problems).

## References

- [1] Bourianoff G 2003 The future of nanocomputing *IEEE Comput.* **36** 44–53
- [2] International Technology Roadmap for Semiconductors 2003

- [3] Goldhaber-Gordon D *et al* 1997 Overview of nanoelectronic devices *Proc. IEEE* **85** 521–40
- [4] Seeman N C 1982 Nucleic acid junctions and lattices *J. Theor. Biol.* **99** 237–47
- [5] Robinson B H and Seeman N C 1987 The design of a biochip: a self-assembling molecular-scale memory device *Protein Eng.* **4** 295–300
- [6] Carbone A and Seeman N C 2002 Circuits and programmable self-assembling DNA structures *Proc. Natl Acad. Sci.* **99** 1–6
- [7] LaBean T H 2003 Introduction to self-assembling DNA nanostructures for computation and nanofabrication *CBGI 2001: Proc. Computational Biology and Genome Informatics (March 2001)* (Durham, NC: World Scientific)
- [8] Reif J H *et al* 2000 Challenges and applications for self-assembled DNA nanostructures *Proc. 6th Int. Workshop on DNA-Based Computers (June 2000)* vol 2054, ed A Condon and G Rozenberg
- [9] Reif J H 2002 DNA lattices: a method for molecular-scale patterning and computation *Comput. Sci. Ind.* **4** 32–41
- [10] Seeman N C 2003 DNA in a material world *Nature* **421** 427
- [11] Huang Y *et al* 2001 Directed assembly of one-dimensional nanostructures into functional networks *Science* **291** 630–3
- [12] Dwyer C *et al* 2002 DNA functionalized single-walled carbon nanotubes *Nanotechnology* **13** 601–4
- [13] Dwyer C 2003 Self-assembled computer architecture: design and fabrication theory *PhD Thesis* University of North Carolina at Chapel Hill
- [14] Dwyer C *et al* 2003 Performance simulation of nanoscale silicon rod field-effect transistor logic *IEEE Trans. Nanotechnol.* **2** 69–74
- [15] Heath J R *et al* 1998 A defect-tolerant computer architecture: opportunities for nanotechnology *Science* **280** 1716–21
- [16] Han J and Jonker P 2003 A defect- and fault-tolerant architecture for nanocomputers *Nanotechnology* **14** 224–30
- [17] Nikolic K *et al* 2002 Fault-tolerant techniques for nanocomputers *Nanotechnology* **13** 357–62
- [18] Ancona M G 1996 Systolic processor designs using single-electron digital circuits *Superlatt. Microstruct.* **20** 461–72
- [19] Ellenbogen J C and Love J C 2000 Architectures for molecular electronic computers: 1. Logic structures and an adder designed from molecular electronic diodes *Proc. IEEE* **88** 386–426
- [20] Fountain T J *et al* 1998 The use of nanoelectronic devices in highly-parallel computing systems *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **6** 31–8
- [21] Goldstein S C and Budiu M 2001 Nanofabrics: spatial computing using molecular electronics *Proc. 28th Annual Int. Symp. on Computer Architecture (July 2001)* pp 178–91
- [22] Peper F *et al* 2003 Laying out circuits on asynchronous cellular arrays: a step towards feasible nanocomputers *Nanotechnology* **14** 469–85
- [23] Porod W *et al* 1993 Quantum cellular automata *Nanotechnology* **4** 49–57
- [24] Gosser K and Pacha C 1998 System and circuit aspects of nanoelectronics *Proc. 24th Conf. on European Solid-State Circuits (Sept. 1998)*
- [25] Niemier M T and Kogge P M 2001 Exploring and exploiting wire-level pipelining in emerging technologies *Proc. 28th Annual Int. Symp. on Computer Architecture (July 2001)* pp 166–77
- [26] Beckett P and Jennings A 2002 Toward nanocomputer architecture *Proc. 7th Conf. on Asia-Pacific Computer Systems Architecture* pp 141–50
- [27] Collier C P *et al* 1999 Electronically configurable molecular-based logic gates *Science* **285** 391–4
- [28] DeHon A 2003 Array-based architecture for FET-based, nanoscale electronics *IEEE Trans. Nanotechnol.* **2** 20–32
- [29] Oskin M *et al* 2002 A practical architecture for reliable quantum computers *IEEE Comput.* **35** (Jan.) 79–87
- [30] Steffen M *et al* 2001 Toward quantum computation: a five-qubit quantum processor *IEEE Micro.* **21** 24–34
- [31] Oskin M *et al* 2003 Building quantum wires: the long and the short of it *Proc. 30th Annual Int. Symp. on Computer Architecture (June 2003)* pp 374–85
- [32] Dwyer C, Johri V, Patwardhan J P, Lebeck A R and Sorin D J 2004 Design tools for self-assembling nanoscale technology *Nanotechnology* **15** 1240–5
- [33] Ng H T, Han J, Yamada T, Nguyen P, Chen Y P and Meyyappan M 2004 Single crystal nanowire vertical surround-gate field-effect transistor *Nano Lett.* **4** 1247–52
- [34] Dwyer C, Vicci L, Poulton J, Erie D, Superfine R, Washburn S and Taylor R M 2004 The design of DNA self-assembled computing circuitry *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **12** (11)
- [35] Mirkin C A, Letsinger R L, Mucic R C and Storhoff J J 1996 A DNA-based method for rationally assembling nanoparticles into macroscopic materials *Nature* **382** 607–9
- [36] Mbindyo J K N, Reiss B D, Martin B R, Keating C D, Natan M J and Mallouk T E 2001 DNA-directed assembly of gold nanowires on complementary surfaces *Adv. Mater.* **13** 249
- [37] Soto C M, Srinivasan A and Ratna B R 2002 Controlled assembly of mesoscale structures using DNA as molecular bridges *J. Am. Chem. Soc.* **124** 8508–9
- [38] Yan H, Park S H, Finkelstein G, Reif J H and LaBean T H 2003 DNA templated self-assembly of protein arrays and highly conductive nanowires *Science* **301** 1882–4
- [39] Audet C and Dennis J E 2000 Pattern search algorithms for mixed variable programming *SIAM J. Optim.* **11** 573–94
- [40] Hough P D *et al* 2000 Asynchronous parallel pattern search for nonlinear optimization *SIAM J. Sci. Comput.* **23** 134–56
- [41] Zitzler E *et al* 2000 Comparison of multiobjective evolutionary algorithms: empirical results *Evol. Comput.* **8** 173–95
- [42] Fieldsend J E and Singh S 2002 A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence *Proc. 2002 UK Workshop on Computational Intelligence (Sept. 2002)* pp 37–44
- [43] Yan H, Feng L, LaBean T H and Reif J H 2003 Parallel molecular computations of pairwise exclusive-or (XOR) using DNA ‘string tile’ self-assembly *J. Am. Chem. Soc.* **125** 14246–7 (Communication)
- [44] Sa-Ardyen P, Jonoska N and Seeman N C 2004 Self-assembly of irregular graphs whose edges are DNA helix axes *J. Am. Chem. Soc.* **126** 6648–57
- [45] Adleman L 1994 Molecular computation of solutions to combinatorial problems *Science* **266** 1021–4